

What we need to do is create an `HTMLElement` abstract class. This class is basically what the `Div` class was before, but made abstract:

```
@dataclass
class HTMLElement(ABC):
    parent: HTMLElement | None = None
    x: int = 0
    y: int = 0

    def compute_screen_position(self) -> tuple[int, int]:
        if not self.parent:
            return (self.x, self.y)
        parent_x, parent_y = self.parent.compute_screen_position()
        return (parent_x + self.x, parent_y + self.y)
```

After that, each HTML element type needs to be a subclass of this class and you can then remove all of the duplication. For example, this is what the `Button` class will now look like:

```
class Button(HTMLElement):
    def click(self) -> None:
        print("Click!")
```

Take a look at what the new version of the code looks like in the attachment to this solution.

There's a couple of interesting things to note:

- The `Div` class is empty, so at the moment its only purpose is to mark that an HTML element is of type `Div`.
- `HTMLElement` is both a dataclass and an abstract base class. `Span` is a dataclass that inherits from `HTMLElement`. As you can see, dataclasses can properly deal with these kinds of inheritance relationships and generate the correct initializer methods.
- As you can see in the new version, you can now also compute screen positions of elements that are not divs, such as the span.

## b) Protocols

Can you use a *protocol* just as easily instead of an abstract base class? What would need to be changed so you can use a protocol class instead of an abstract base class?

### Solution

The issue that you're going to run into is that with protocols you rely on structural typing (duck typing) instead of inheritance. That means that if you have an `HTMLElement` protocol and you define properties and methods inside of it, the other classes won't be able to use that if they don't inherit from `HTMLElement` anymore. There are a couple of ways to fix this:

- Even though protocols use structural typing, you can still inherit from them, just like with abstract base classes. You may think that's the best solution, but I personally prefer to keep structural typing 'pure' so that when you use protocols in your code, you know that you never have to inherit from them. If you still need to inherit from protocols in some cases, it may result in confusion: when do we need to inherit from them and when not?
- You could create two layers of inheritance. At the top layer is a protocol class that contains only the `compute_screen_position` method and properties for the parent, x, and y. You could then have an abstract base class (named `HTMLElementExceptionBase` for example), that follows the protocol's structure and then each specific HTML element type would then be a subclass of `HTMLElementExceptionBase`. In the areas of your application where you want to rely on structural typing, you can then use the `HTMLElementException` protocol class, while still being able to provide a basic implementation of parents and positioning for the specific HTML elements.